Date: 27/07/2024

WEEK 2 : Optimization Algorithms

⟹ Batch v/s Mini-batch gradient desent

$$X = [x^{(1)} \quad x^{(2)} \quad \cdots \quad x^{(m)}]$$

(n_x, m)

$$Y = [y^{(1)} \quad y^{(2)} \quad \cdots \quad y^{(m)}]$$

$(1, m)$

What if m = 5million (5,000,000)

mini-batches of 1000 each

$$X = [\underbrace{x^{(1)} \text{ to } x^{(1000)}}_{X^{\{1\}}} \mid \underbrace{x^{(1001)} \text{ to } x^{(2000)} \cdots}_{X^{\{2\}}} \quad \underbrace{\quad}_{X^{\{5000\}}}]$$

$X^{\{1\}}$ $(n_x, 1000)$ $\quad$ $X^{\{2\}}$ $(n_x, 1000)$ $\cdots$ $X^{\{5000\}}$

Similarly for Y too;

$(1, 1000)$

Date:

Mini-batch t : $X^{\{t\}}, Y^{\{t\}}$

Mini-batch gradient descent :

Vectorised:

for $t = 1, \ldots, 5000$

{

Forward prop on $X^{\{t\}}$

$$Z^{[1]} = W^{[1]} X^{\{t\}} + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

⋮

$$A^{[L]} = g^{[L]}(Z^{[L]})$$

$X^{\{t\}}, Y^{\{t\}}$

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^{l} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{l} \|W^{[l]}\|_F^2$

Regularisation

Date:

Backprop to compute gradients wrt $J^{[t]}$ (using $X^{\{t\}}, Y^{\{t\}}$)

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha db^{[l]}$$

}

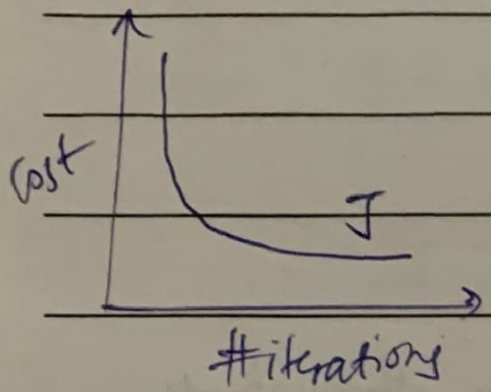
"1 epoch" → 1 pass through the training set

When using a large dataset; the mini-batch is works much faster.

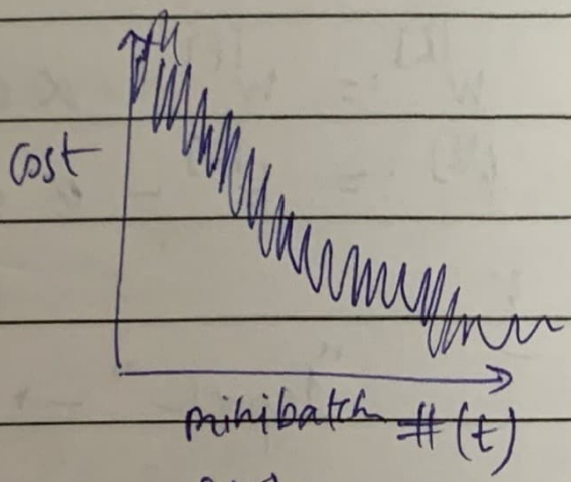
Date:

→ Training with mini-batch gradient descent:

Batch gradient descent



Mini-batch grad. descent



$J^{\{t\}}$ using $(X^{\{t\}}, Y^{\{t\}})$

Not strictly decreasing, but should trend downwards

→ Choosing mini-batch size:

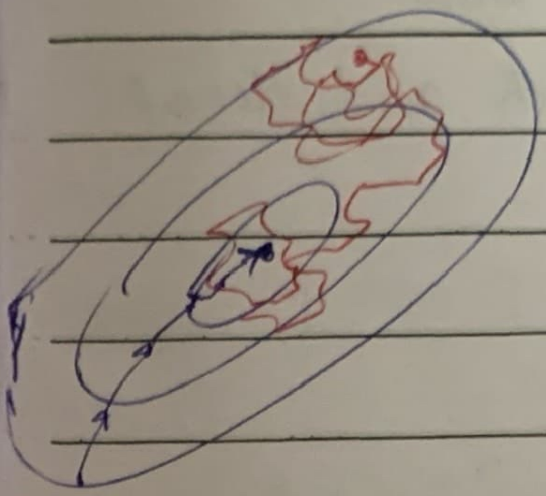
mini batch size = m; Batch grad. descent

$X^{\{1\}}, Y^{\{1\}} = X, Y$

size = 1; Stochastic grad. descent

Every example is its own mini-batch.

Red for Stochastic (doesn't stop at minimum always around minimum)
Blue for batch
Date:



In practice:

size in both

1 and m.

① Batch grad. desent.

Too long per iteration

② stochastic grad. descent

loosing all speed from vectorisation

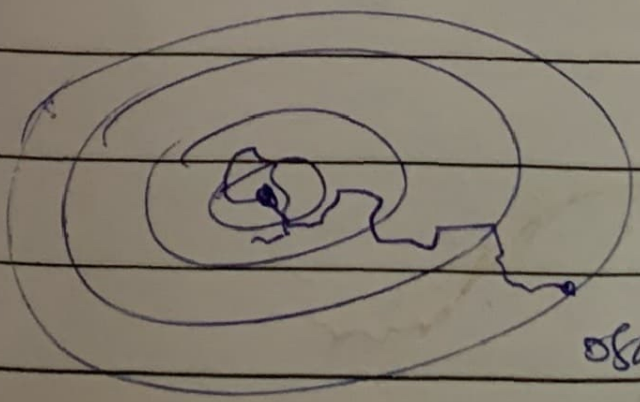
∴ each example is processed : inefficient

③ ~~In~~ both 1 and m:

not too big/small

→ fastest learning → vectorisation ✓

→ make progress without processing the entire training set



not exactly at minimum (converge) but can be oscillated very close to minimum by reducing learning rate

Date:

① small training set: Batch grad. descent.
($m \leq 2000$)

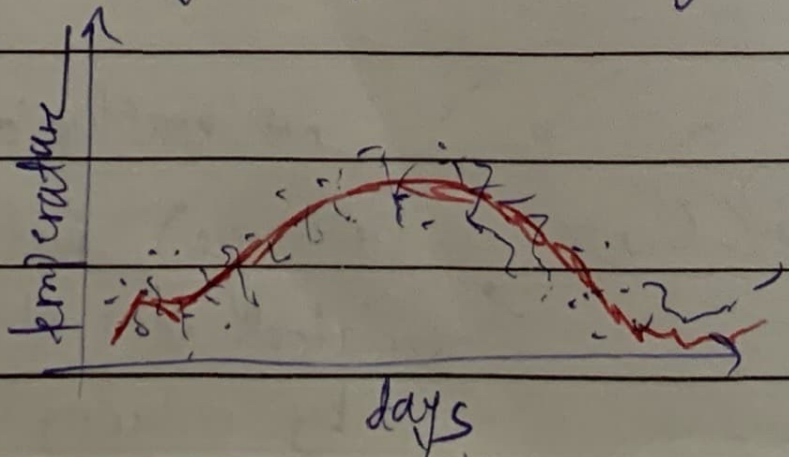
② Typical mini batch sizes:

64, 128, 256, 512 ...
2^6 2^7 2^8 2^9 (power of 2)
common
∵ how the computer memory is organised

Make sure mini-batch size fits
~~its~~ in CPU/GPU memory

⇒ Different optimization Algorithms:

① Exponentially weighted (moving) averages:



Date:

$$V_t = \beta \cdot V_{t-1} + (1-\beta)\theta_t$$

$\beta = 0.9$: $\approx$ 10 day temp.

$\beta = 0.98$: $\approx$ 50 days temps.

V_t : approximately averaging over $\approx \frac{1}{1-\beta}$ days' temperature

as $\beta \uparrow$ days $\uparrow$: .

smooth curves.

$\beta = 0.5$: $\approx$ 2 days

: very noisy, very quickly adapting to temperatures

$v_{100} = 0.9 v_{99} + 0.1 \theta_{100}$

$v_{99} = 0.9 v_{98} + 0.1 \theta_{99}$

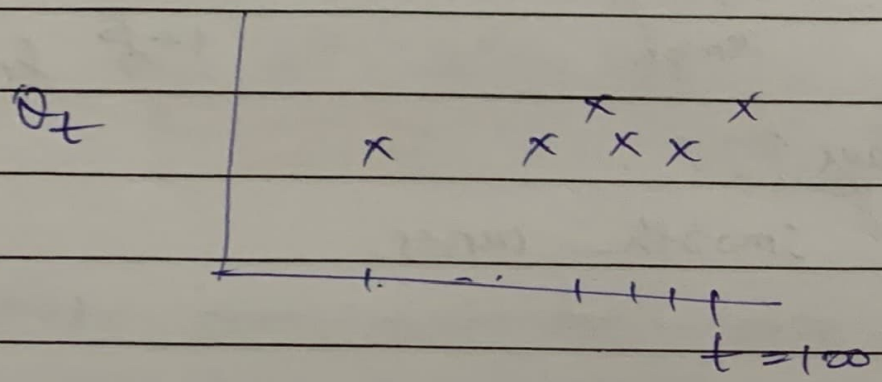
$\vdots$

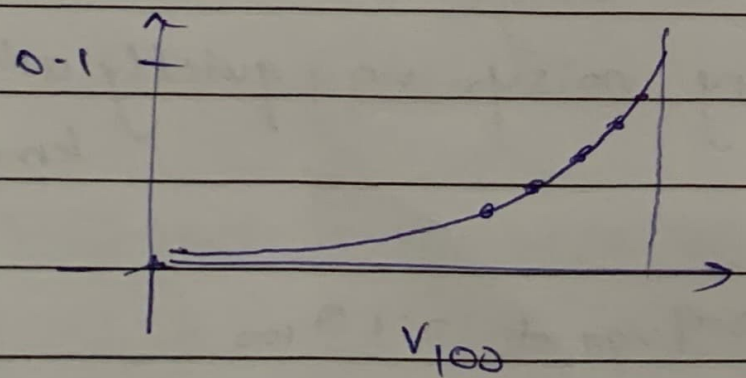
$v_{100} = 0.1\theta_{100} + 0.9 v_{99}$ ... : continues.

$v_{99} = 0.1\theta_{99} + 0.9 v_{98}$

Date:

$$V_{100} = 0.1\theta_{100} + 0.1 \times 0.9\,\theta_{99} + 0.1 \times (0.9)^2 \theta_{98} + 0.1 \times (0.9)^3 \cdot \theta_{97} + \cdots$$





exponentially decaying function

$$0.9^{10} \approx 0.35 \approx \frac{1}{e}$$

Date:

$\underbrace{(1-\varepsilon)}_{0.9}^{1/\varepsilon} = \frac{1}{e}$ means focusing on only 10 days.

$0.98^{50} \approx 1/e$ $\therefore$ 50 days when $\beta = 0.98$

Implementation:

$v_0 = 0$

$v_1 = \beta v_0 + (1-\beta)\theta_1$

$v_2 = \beta v_1 + (1-\beta)\theta_2$

⋮

$v_0 := 0$

$v_0 := \beta v + (1-\beta)\theta_1$

$v_0 := \beta v + (1-\beta)\theta_2$

Date:

$V_\theta = 0$

Repeat {

Get next θ_t

$V_\theta := \beta V_\theta + (1-\beta)\theta_t$

}

⇒ Bias correction:

Change the formala

$$\frac{V_t}{1-\beta^t}$$

$\because \beta \uparrow \quad (1-\beta^t) \downarrow$

$\therefore V_t \uparrow$ ∴ at first two days also it's a good estimate.

Date:

(2) Momentum / Gradient descent with momentum

(works faster than batch grad. descent)

Momentum:

$V_{dw} = 0$; $V_{db} = 0$ (initialised to matrix of zeros)

on iteration t:

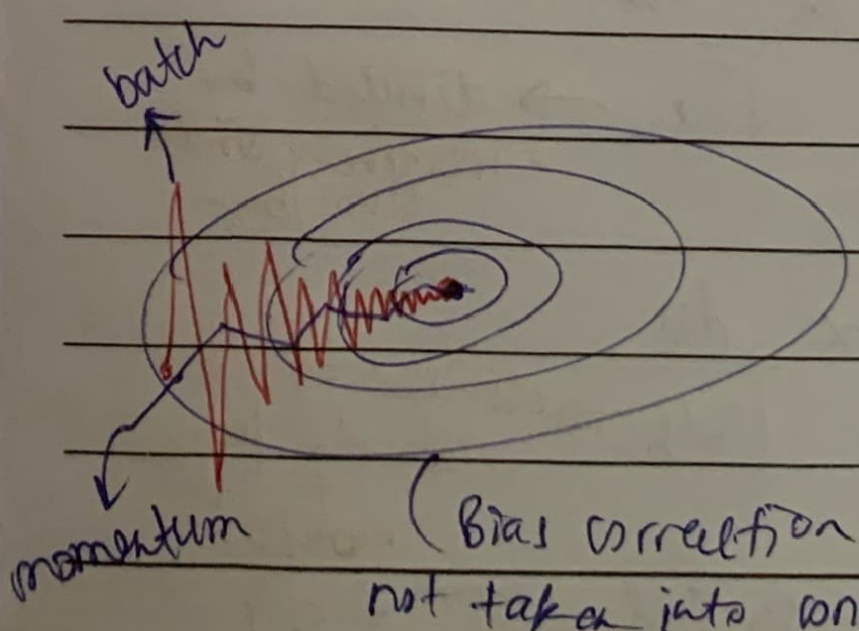
Compute dw, db on current mini-batch:

$V_{dw} = \beta V_{dw} + (1-\beta)\, dw$

$V_{db} = \beta V_{db} + (1-\beta)\, db$

$w := w - \alpha V_{dw}$

$b := b - \alpha V_{db}$



Hyperparameters: α, β

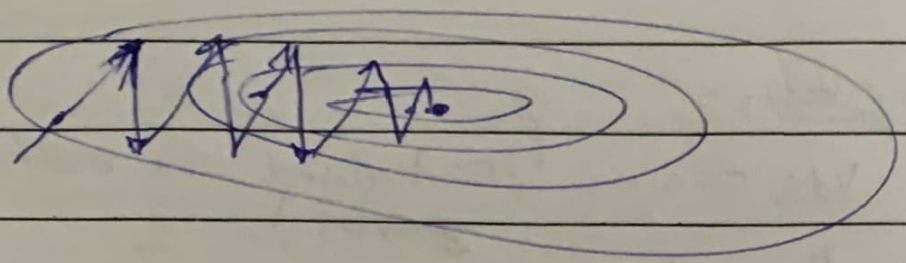
$\beta = 0.9$

≈ 10 previous gradients average

(Bias correction not taken into consideration)

Date:

(3) RMS prop
(root mean squared prop)



On iteration t:

Compute dw, db on current mini-batch

element-wise

$$S_{dw} = \beta_2 S_{dw} + (1-\beta_2) dw^2$$
$$S_{db} = \beta_2 S_{db} + (1-\beta_2) db^2$$

$$w := w - \alpha \frac{dw}{\sqrt{S_{dw}}}$$

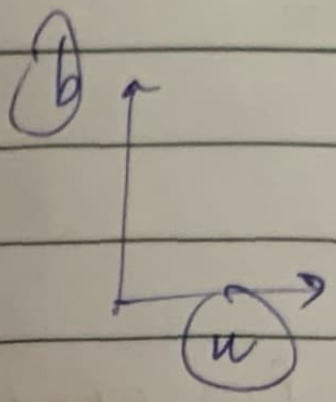
$\sqrt{S_{dw}}$ → divided by a relatively small number

$$b := b - \alpha \frac{db}{\sqrt{S_{db}}}$$

$\sqrt{S_{db}}$ → divided by a relatively large number

Note: $S_{dw} + \varepsilon$ and $S_{db} + \varepsilon$ is used so that denr $\neq 0$

Date:



want b ↓ and w ↑

= that is happening by RMS prop.

A higher learning rate can then be used, for faster learning.

(4) Adam optimization algorithm:

Adaptive moment estimation

$V_{dw}, S_{dw}, V_{db}, S_{db} = 0$;

On iteration t:

Compute dw, db using current mini-batch

$V_{dw} = \beta_1 V_{dw} + (1-\beta_1) dw$

$V_{db} = \beta_1 V_{db} + (1-\beta_1) db$

} "momentum" β_1

$S_{dw} = \beta_2 S_{dw} + (1-\beta) dw^2$

$S_{db} = \beta_2 S_{db} + (1-\beta) db^2$

} "RMS prop" β_2

Date:

$$V_{dw}^{corrected} = V_{dw} / (1 - \beta_1^t)$$

$$V_{db}^{corrected} = V_{db} / (1 - \beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw} / (1 - \beta_2^t)$$

$$S_{db}^{corrected} = S_{db} / (1 - \beta_2^t)$$

} Bias correction

$$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corr} + \varepsilon}}$$

$$b := b - \alpha \frac{V_{db}^{correct}}{\sqrt{S_{db}^{corr} + \varepsilon}}$$

Commonly used; very effective;

Hyperparameters:

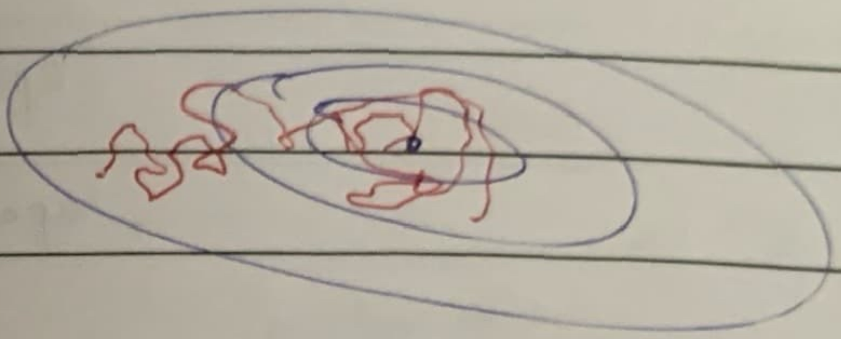
α: needs to be tuned

β_1: 0.9 (dw) value β_2: 0.999 (dw^2) value

$\varepsilon = 10^{-8}$

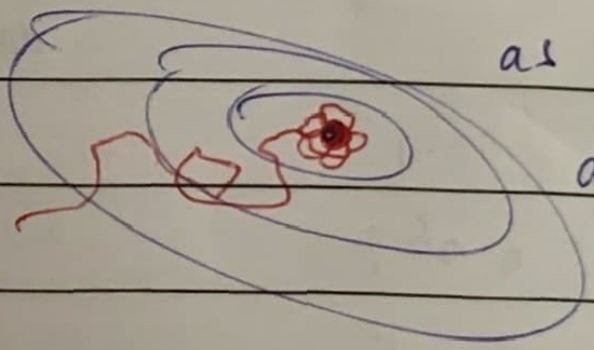
Date:

⇒ Learning rate decay :



While using mini-batches :

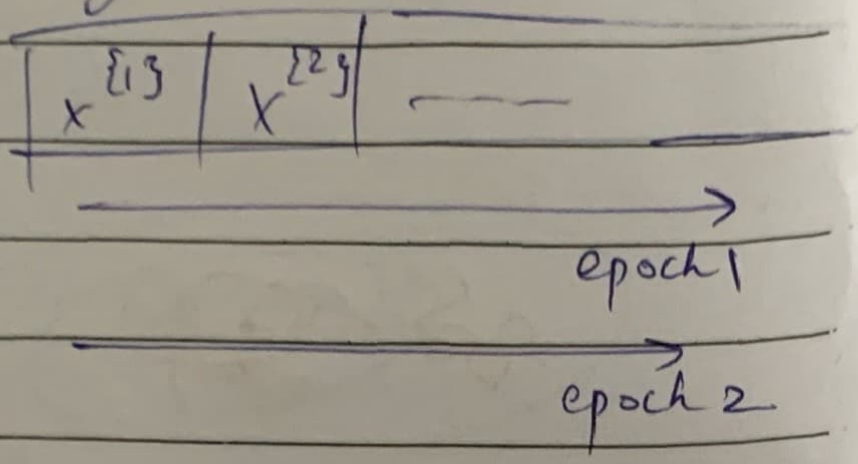
After slowly reducing of α :



as $\alpha \downarrow$, it converges almost and surrounds very near the minimum.

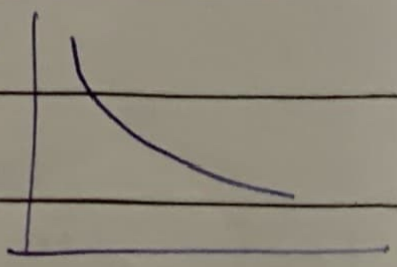
Date:

1 epoch = $\frac{1}{1}$ pass through data



$$\alpha = \frac{1}{1 + \text{Decay-rate} * \text{epoch-Number}} \alpha_0$$

Epoch	α	$\alpha_0 = 0.2$
		decay-rate = 1
1	0.1	
2	0.067	
3	0.05	
4	0.04	
⋮	⋮	



Date:

other methods: (for decay rate):

→ $\alpha = 0.95^{\text{epoch\_number}} \cdot \alpha_0$

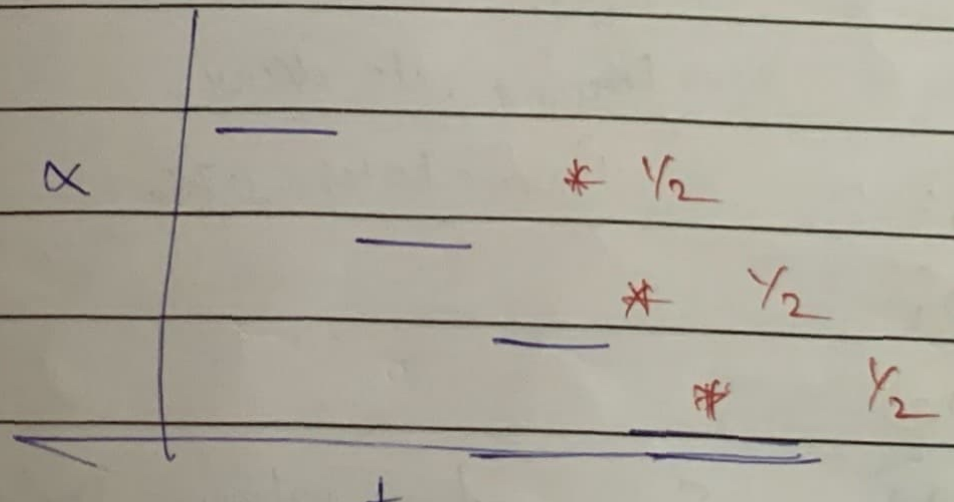
(exponentially decay)

→ $\alpha = \frac{k}{\sqrt{\text{epoch\_number}}} \cdot \alpha_0$ or $\frac{k}{\sqrt{t}} \cdot \alpha_0$

k → constant

t → mini-batch number

→



discrete staircase

→ Manual decay